This section contains items that are required to get the patent process under way

Document Author (name and email):

Title of Invention: Active Server Pages for Hotmail (ASPH) **MS File # (if known):**

Inventor(s): Baskaran Dharmarajan (baskd), Vikram Sardesai (vikrams)

Introduction: Many ISAPI based applications that generate formatted responses (in HTML, XML, WML etc.) need a template system. The template system is needed for two reasons – 1) developers working on presentation languages like HTML vary significantly in skill set from developers working on ISAPI applications, which usually involve the C/C++ languages. If the output were directly generated from the ISAPI "C/C++" code, it would make it cumbersome for the two set of developers to work. 2) UI elements change far rapidly than the basic logic such as the access of data elements from the backend and the business logic. So, separating the presentation from the business logic is vital to a maintainable web application. ASPH is Hotmail's UI template system.

The ASPH runtime can execute any file in the ASPH file format. The file format contains a language table, one file index per language and the files as a bunch of code sections. The code sections contain the instructions to display UI based on the ASPH instruction set. A compiler converts the UI templates into an ASPH file. While other languages can be compiled into the ASPH file format, Hotmail uses a language called ASPL, which is similar in syntax to Microsoft ASP. Hotmail UI developers develop their UI templates in ASPL and then compile it into the ASPH file. This file is then executed by the runtime system at the behest of the Hotmail ISAPI application to render the appropriate HTML/XML/WML pages.

Strategic Importance: The ASPH implementation provides a unique way of developing the UI for a web application employing ISAPI extensions. It allows a separation of the UI from the business logic, which is contained in the ISAPI extension. Standard ways of developing UI are by using ASP, ASP.NET, JSP etc. However, these do not yield the same performance that ISAPI does. There is currently no standard way of developing a UI subsystem when using ISAPI for web application development. This concept could be extended to offer an out-of-the-box UI development system for use with ISAPI. The file format and the associated instruction set is designed for best performance with the least amount of complexity.

Note: While we use Microsoft ISAPI to describe a class of high performance web applications, note that the need and the circumstances of any application based on similar technologies such as Netscape NSAPI, Apache CGI and any such basic web server to application binary interfaces.

Description of the invention:

In a nutshell, the ASPH subsystem comprises of the three core components, the ASPL pages, the execution engine and the ASPH compiler (called as CASPH). The ASPH compiler compiles the ASPL pages into a proprietary byte code format, the end result of which is a single ASPH file. The execution engine loads the ASPH file at runtime and executes the byte code for the selected file.

Following is a detailed description of the ASPH system.

ASPL Pages

These pages contain the html/XML/WML that renders the UI mixed with the presentation logic. Since the objective is to be able to generate dynamic web pages, they contain ASPH code snippets, which are executed at runtime to dynamically generate the resulting html that is finally sent back to the client.

The html code is any valid html; it is text from the ASPH compiler's perspective. The ASPH code is enclosed within the <% and %> delimiters. The constructs supported are more or less on the lines of the constructs in actual Microsoft ASPL code. Following is a list of some of the constructs allowed in

ASPH code.

1. SET: This allows an ASPH variable to set to a value.

   ```
   <% set varname  value %>
   ```

   varname: name of the variable
   value: value of the variable, which can be text or a concatenation of one or more variables and text. Variables referred in the value portion should be enclosed within ${ and }.
   e.g.

   ```
   <% set TitleText MSN Hotmail %>
       <% set titlelink ${server}/cgi-bin/quiklist?${usermagic} %>
   ```

2. INCLUDE: This allows an ASPL file to include another ASPL file. This is to allow reuse of UI components separated into individual files and then used wherever needed.

   ```
   <% include filename %>
   ```

   filename: name of the file to be included
   e.g.

   ```
   <% include topstuff.asp %>
   ```

3. IF/ELSE/ELSIF: This allows the conditional execution of code.

   ```
   <% if  expression1 %>
    ....
    ....
   <% elsif expression2 %>
   ....
   ....
   <% elsif expression3 %>
   ....
   ....
   <% endif %>
   ```

   expression1, expression2 and expression3 are logical expressions composed of ASPL variables and relational operators (==, !=) and logical operators (&&, ||).
   e.g.

   ```
   <% if  Alpha == ${Beta} %>
        <% set title SomeTextHere %>
     <% elsif Alpha == ${Beta} && Alpha != ${Gamma} %>
        <% set title SomeTextHere %>
     <% elsif Alpha == alphatext && (Alpha == ${Theta} || Beta == ${Gamma} ) %>
        <% set title SomeTextHere %>
     <% endif %>
   ```

The variable type system

Asph uses indices instead of names so that every lookup is an array indexing. Since there are many classes of variables, the following types of indices are identified.

The basic idea is that the ISAPI code can use these indices to set and get values for the specific variable. The ASPL compiler uses the indices table declared as enumerations in the C header files as data and derives the names out of them by basically removing the prefix. Thus, cAsphCabc is equivalent to the variable name "abc" to the ASPL compiler.

Where a given variable is entirely internal to ASPL, there is no need for an explicit index that is exposed to the C code. These variables are assigned indices in the order in which they are encountered by the ASPL compiler.

Where a given variable is used in the ASPL code before they are set, unless they are built in or Site configuration variables (in which case they have a default value), the value is derived from the URL parameters. This feature will be deprecated in future, forcing the setting of variables before they are used.

<table>
<tr><th>Type</th><th>Comment</th></tr>
<tr><td>Built-In</td><td>Variables that are supported by the base Asph Runtime. These will always have a value, even though specific ISAPI code can set them to a different value than the default one supplied by the runtime.
<table>
<tr><td>Examples</td><td>ImageServer, IsIE5</td></tr>
<tr><td>C constant</td><td>cAsphBImageServer, cAsphBIsIE5</td></tr>
<tr><td>Declared In</td><td>AsphSymbolTable.h</td></tr>
<tr><td>Set In</td><td>AsphBuiltIn.cpp</td></tr>
</table></td></tr>
<tr><td>ISAPI</td><td>Variables that don't have any default value unless the specific ISAPI that is printing the ASPL file sets one. If there is no value, the runtime may search in the URL parameters for a value and use it. But this URL Items search may be removed in future and is deprecated even now.
<table>
<tr><td>Examples</td><td>HasAttachements, RichTextYes</td></tr>
<tr><td>C constant</td><td>cAsphCHasAttachments, cAsphCRichTextYes</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>Declared In</td><td>AsphSymbolTable.h</td></tr><tr><td>Set In</td><td>Various CGIs</td></tr></table></td></tr>
<tr><td>Internal</td><td>Variables that are declared and consumed with ASPL files. Note that the variable may be set in one ASPL file and used in another file that may include the first file.<table><tr><td>Examples</td><td>KillNoRadioSelected, lameheliumworkaround</td></tr><tr><td>C constant</td><td>None</td></tr><tr><td>Declared In</td><td>No where</td></tr><tr><td>Set In</td><td>ASPL files</td></tr></table></td></tr>
<tr><td>HRS Static Language Based</td><td>Static HRS (HRS is the Hotmail resource System that is used for localization) variables that are based on only language.<table><tr><td>Examples</td><td>S01, S02</td></tr><tr><td>C constant</td><td>None</td></tr><tr><td>Declared In</td><td>No where</td></tr><tr><td>Set In</td><td>s.hrs</td></tr></table></td></tr>
<tr><td>HRS Dynamic Language Based</td><td>Dynamic HRS variables that are based on only language.<table><tr><td>Examples</td><td>D01, D02</td></tr><tr><td>C constant</td><td>None</td></tr><tr><td>Declared In</td><td>No where</td></tr><tr><td>Set In</td><td>d.hrs</td></tr></table></td></tr>
<tr><td>HRS Static Locale Based</td><td>Static HRS variables that are based on both language and country.</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>Examples</td><td>S11, S12</td></tr>
<tr><td>C constant</td><td>None</td></tr>
<tr><td>Declared In</td><td>No where</td></tr>
<tr><td>Set In</td><td>s.hrs</td></tr>
</table>
</td></tr>
<tr><td>HRS Dynamic Locale Based</td><td>Dynamic HRS variables that are based on both language and country. E.g.
<table>
<tr><td>Examples</td><td>S11, S12</td></tr>
<tr><td>C constant</td><td>None</td></tr>
<tr><td>Declared In</td><td>No where</td></tr>
<tr><td>Set In</td><td>d.hrs</td></tr>
</table>
</td></tr>
<tr><td>Site Configuration</td><td>Variables that represent the site configuration values. These variables cannot be set to any other value than the ones in the site configuration, unlike ISAPI variables.
<table>
<tr><td>Examples</td><td>SiteConfig::ABCSMigrationCompleted,
SiteConfig::EFormsLinkServer</td></tr>
<tr><td>C constant</td><td>cAsphSiteConfig__ABCSMigrationCompleted,
cAsphSiteConfig__EFormsLinkServer</td></tr>
<tr><td>Declared In</td><td>settings_asph.h</td></tr>
<tr><td>Set In</td><td>settings_asph_inline.h</td></tr>
</table>
</td></tr>
<tr><td>File Name</td><td>Variables that represent File names. Used in include statements and by ISAPI code to print specific files.
<table>
<tr><td>Examples</td><td>Attach.asp, hotmail.js</td></tr>
</table>
</td></tr>
</table>

	C constant	**cAsphFAttach, cAsphFHotmail__js**
	Declared In	**AsphFileTable.h, casph.config (See WAIT:)**
	Set In	**Automatic**
Internal Built-In	Variables set by such files that are part of the standard list that are pre-included based on the request characteristics.	
	Examples	SmallMSNLogoImage, WebMasterAcct
	C constant	**None**
	Declared & Set In	**ASPL files such as hotmail.asp**

The Instruction set

The execution engine, which is the ISAPI runtime code that deals with response page generation, basically executes a single ASPH file, that contains, among other things, the byte code based instructions for the various ASPL files. The ASPL files are compiled by the ASPL compiler into byte codes, each a byte long, followed by zero or more arguments. The execution engine also offers a few "registers." The program counter register tracks the offset of the next instruction. The file start register is used to keep track of the file start locations so that relative offsets could be used. A stack is used into which the value from the program counter and file start are pushed into and popped from, when files are included from other files. The CompareResult register stores the result of the most recent comparison operation. The definition of the flag bits are:

```
#define cCompareResultEqual        0x1 // bit 1
#define cCompareResultNotEqual     0x2 // bit 2
#define cCompareResultEmpty        0x4 // bit 3
#define cCompareResultNotEmpty     0x8 // bit 4
```

The various byte codes and their meanings are discussed in the table below.

Cmd	Code	Arguments			What the execution engine does
		Name	Size (bytes)	What does it mean?	
Text	t	Length	2	The length of the text to be printed out	Reads in the 2 bytes of length. Then it simply memory copies that many bites on to the output. Contrast this against what ASPL has to do – it has to copy character by character to the output, as it is looking for the angle brackets in between.
		Text	Variable	The text to be printed out as is	
Print	p	Interpretation/Transformation	1	This argument specifies how to index should be interpreted. See the interpretation/transformation table below on details.	Prints the value of the variable pointed to by the index by looking it up from the value table. This is the equivalent of <%=%> construct.
		Index	4	The index of the variable whose value is to be printed.	
coMpare	m	Index1	4	The index of the first variable	Compares two variables and updates the CompareResult register.
		Index2	4	The index of the second variable	
Jump	j	When	1	Gives the flags on which to jump	If any of the flags set is also set in the CompareResult register, the execution jumps to the offset in question and starts executing the instruction at that offset by loading the program counter with the offset
		Offset	4	The relative address in the compiled file to	

				jump to. The offset of the very first instruction in a compiled file is assumed to be zero.	in question. If the "When" parameter is 0, this is an absolute jump. Otherwise, the compiler treats the instruction as a "NO OP" and just continues with the next instruction code.
callBack	b	Iteration	4	The index of the variable that holds the iteration count.	This is for the for loop implementation as well.
		Key	4	The index of the variable that holds the key.	
		Index	4	The variable that holds the pointer to the call back function to invoke	
		Length	4	The length of the statements with which the call back needs to be executed	
		Code	variable	The instruction code that needs to be executed by the call back ending with a quit instruction.	
Load	1	Index	4	The index of the variable into which the load is going to be performed	If no dereferencing is needed, this is equivalent to setting a variable to a literal string or numerical value. Otherwise this is equivalent to an assignment statement where one variable value is assigned to another. Note that where the value is a number, an index could be loaded into another, even though there is no explicit statement
		Interpretation/Transformation	1	This argument specifies how to index should be interpreted. See the interpretation/transformation table below on	

				details.	signifying it.
		Length	4	Length of the value that is going to be loaded.	
		value	Variable	The value	
Examine	x	Index	4	The index to be examined.	The value pointed to by the index is examined and the empty bit is set in the CompareResult register if it is an empty string
Equal	e	Index	4	The index of a variable	Test the value of the variable with the given index against the literal value. If they are equal, set the cCompareResultEqual bit. Otherwise set the cCompareResultNotEqual bit in the CompareResult register.
		Length	4	Length of the value to be tested against.	
		value	Variable	The literal value	
Add	a	Amount	4	The amount to be added.	The value pointed to by the index is converted into a number, the amount added and the value is set to point to the string representing the sum.
		Index	4	The index of the variable to add the amount to.	
Call	c	Index	4	The index in the file table where the offset of the starting instruction corresponding to a given ASPL file is stored.	The equivalent of include. Direct offsets are not used to allow Just In Time(JIT) compilation for the debug mode. The execution engine pushes the offset of the next instruction into the stack and jumps to the offset pointed to by the file table at the specified index.

		Interpretation/Transformation	1	This specifies how the index should be interpreted.	If 0, it directly indexes into file table, else it dereferences the index to get the file index and then indexes into the file table.
Start	s	How	1	1 if directly invoked and 0 if not. This only describes how the compiler first encountered the file. If 1, the file ends with a quit instruction. Otherwise, return.	For diagnostic/reverse compiler purposes.
		Index	4	The file index	
		Length	2	The length of the file.	
Switch	w	Index	4	The index of the variable whose value is switched upon. This index is expected to point to another index.	For dealing with internal built in variables.
		Length	2	Length of the offset array.	
		OffsetArray	Variable	The list of index value-offset value	

				pairs. If the value in the switch index is equal to the index value of the pair, the execution jumps to the specified offset.	
Return	R				The stack is popped and the offset stored there is loaded into the program counter.
Quit	q				Stops interpreting any further instructions.

Bit	Name	Meaning
0	Dereference	If 0, the value is a literal and can be loaded directly. Otherwise, the value contains an index. The value of the contained index should be operated on.
1-3	Transformation	000 – none 001 – URL encode 002 – HTML encode
4	Concatenate	Concatenate to the value of the index being loaded into. Meaningful only for the load instruction. If 0, the load will assign this as the new value. Otherwise, it will do the interpretation and then concatenate the resultant value to the existing index value.

Interpretation/Transformation

Special Situations

There are many special situations that are encountered by the Asph system. These are mostly dealt with by using built in callbacks. These are described in the following sections.

In addition to the callbacks below, internal built in variables are dealt with using the switch statement. Basically, in ASPL, AsplInclude files such as hotmail.asp were printed before any ASPL file was printed once per each ISAPI in the ASPL constructor, whether needed or not. Also, these days these files contain so many set statements, some of which are for variables that are used in one or two places and others that may be positively dead. So, in Asph, these variables are classified as internal built in, meaning, these variables derive their built in default value from one of the AsplInclude files. So, hotmail.asp is compiled with a switch statement at the top. Based on which variable is being used in the ASPL page, if the variable is an internal built in variable, the runtime executes the appropriate AsplInclude file, such as hotmail.asp after loading the index of the variable being loaded into cAsphBInternalBuiltInVariableToSwitchOn. The switch statement switches on this variable and in each case takes to the offset where the load statement for the variable whose default value is being looked up resides. The

ASPL compiler inserts a quit after every load statement so that, the file execution of hotmail.asp ends after the loading of the default value into the internal built in variable. Note that once loaded, the file doesn't have to be executed again for the same variable, as the value is loaded into the internal variable tables of the Asph runtime.

Call back	Parameter	What it holds	Comments
ISAPI Callback	Iteration	The index of the variable representing the iteration of the FOR loop.	When you have something like: <% for folder in folders %> <%=folder%> <%endfor%> The index of the variable "folder" goes into Iteration. The string "folders" is pointed to by Key. And the code length is the length of the print statement to print the value of count, namely, 6. Note that the actual call back for the loop has to be set by the ISAPI code by the SetCallback() call. Also, once called, the call back logic may do whatever it deems it needs to do.
	Key	The key string.	
	Call back Index	cAsphCCallback	
	Code Length	The statements included within the FOR loop.	
	Code	The compiled byte codes for the statements included within the FOR loop.	
Range Callback	Iteration	The index of the variable representing the iteration count of the FOR loop.	When you have something like: <% for count in Range:10;-1;1 %> <%=count%> <%endfor%> The index of count goes into Iteration. The string "Range:10;-1;1" is pointed to by Key. And the code length is the length of the print
	Key	The Range string of format: "Range: Start;Increment/Decrement;End"	
	Call back Index	cAsphBRangeCallback	

			statement to print the value of count, namely, 6.
	Code Length	The statements included within the for loop.	
	Code	The compiled byte codes for the statements included within the FOR loop.	
HRS Range Set up Callback.	Iteration	Don't care.	Since, HRS compound strings may contain variable names, whose indices cannot be known until the ASPL files are compiled, the ASPL compiler checks with HRS and converts the variable names to indices. Since the HRS file cannot be changed at that point, the compiler puts in this call to load the variable indices (by order). The HRS compound strings refer to variables by ordinal numbers (in the order they appear in the English string).
	Key	Don't care.	
	Call back Index	cAsphBHrsOrdinalSetupCallback	
	Code Length	The length of the HRS ordinal array.	
	Code	The compiled byte codes for the statements included within the FOR loop.	
Variable Filename Callback	Iteration	The index of the variable that will hold the resultant file index	Mainly to facilitate xincludes, the ASPL compiler uses the variable file name callback to get the index of a name such as wc_${_lang}${country}.asp. After loading a variable with the string representing the pattern wc_ followed by the user's language and country and ending with .asp, the compiler
	Key	The index of the variable that holds the file name as a string.	
	Call back Index	cAsphBVariableFileNameCallback	
	Code Length	0	

<table>
<tr><td></td><td></td><td></td><td rowspan="2">inserts a call to look up the file index for the name and includes that index.</td></tr>
<tr><td></td><td>Code</td><td>None</td></tr>
<tr><td rowspan="5">Atoi Callback</td><td>Iteration</td><td>The index of the variable that will hold the resultant value.</td><td rowspan="5">This is used to implement the <%=$var%> construct. In this case, var is loaded with the index of the variable it contains in the string form, such as "33777743." When the derefencing needs to happen, the ASPL compiler inserts an atoi callback to get the integer value of the string, and use it as a variable index to lookup the value, which is set into variable represented by the iteration parameter.</td></tr>
<tr><td>Key</td><td>The index of the variable that holds the index number as a string.</td></tr>
<tr><td>Call back Index</td><td>cAsphBAtoiCallback</td></tr>
<tr><td>Code Length</td><td>0</td></tr>
<tr><td>Code</td><td>None</td></tr>
<tr><td rowspan="4">Pipe Callback</td><td>Iteration</td><td>The index of the variable that will hold the resultant value.</td><td rowspan="4">This is used to implement the <%=|var%> construct. In this case, var is loaded with the index of the variable it contains in the string form, such as "33777743|33777748" When the derefencing needs to happen, the ASPL compiler inserts an pipe callback to get the integer value of the string, and use it as a variable index to lookup the value, which is set into variable represented</td></tr>
<tr><td>Key</td><td>The index of the variable that holds the list of pipe separated index numbers as a string.</td></tr>
<tr><td>Call back Index</td><td>cAsphBPipeCallback</td></tr>
<tr><td></td><td></td></tr>
</table>

	Code Length	0	by the iteration parameter as a list of string values each separated by a new line.
	Code	None	
Lookup Variable Name Callback	Iteration	The index of the variable that will hold the resultant variable value.	The variables wcid and soid are used both ways, namely as <%=wcid%> and <%=$wcid%>. Therefore, the atoi callback cannot be used to dereference <%=$wcid%>. So, this callback was invented. Here, we basically lookup the index of the name contained in wcid (usually a wc content provider name) and set the value of that variable name into the iteration parameter, which the ASPL compiler can display with additional code. Wcid and soid are the only special cases that need support this way at this time.
	Key	The index of the variable that holds the name of the variable to lookup.	
	Call back Index	cAsphBLookupVarNameCallback	
	Code Length	4	
	Code	The index type to lookup, currently cAsphCwcid and cAsphCsoid supported.	

CASPH

This is the ASPH compiler. Simply stated, it translates the ASPH code in the ASPL files to ASPH byte code and finally generates the a single ASPH file usually called (by convention) i.asph. CASPH maintains symbol tables and file tables to maintain a mapping of names to indices and to finally put this information into the i.asph header and body.

The following are the phases in the compilation process:

<u>Setup:</u> In this phase, the compiler will load the file table and symbol table with the files and variables which are known at compile time. AsphFileTable.h is

a C header file, that specifies the list of files that the ISAPI code is interested in. The compiler will extract the ASPL file names from this file and then load the file information into the file table. Similarly, it will load the ISAPI variable information from AsphSymbolTable.h.

Compile: As mentioned earlier, the compiler needs to compile ASPL files for each language supported.

Link: In this phase, the compiler will basically link together the component main[LANG].asph files and generate the i.asph file with the appropriate file headers.

A few cases of the translation from ASPL code to ASPH byte codes are discussed below to illustrate how the system works.

1. **Set**: A set is translated into one or more Load instructions. If the value that the variable is being set to is text, then only one Load instruction is generated. However, if the value is a concatenation of text and other variables, then a series of Load instructions are generated with appropriate arguments indicating how to concatenate the parts and finally load the resulting value into the destination variable.

2. **Text**: A stream of text (this could be just text or html code, basically anything other than ASPL code) is translated into a text instruction with appropriate arguments. The compiler buffers text till it sees something that is not text and at that point generates the text instruction. So even though the compiler processes the file line by line, it will generate only 1 text instruction for a stream of text.

3. **If/Else**: This is translated into a combination of Compare-Jump or Expression-Jump instructions depending on the type of expression being tested in the if condition. If the expression is a simple expression, meaning not involving logical operators && and ||, it will be translated into Compare-Jump. The Compare tests the expression and the jump will take care of jumping to the right location depending on the Boolean value of the expression.

   If the expression is a compound expression, then the compiler will generate an expression tree, which is basically a binary tree. It then generates the Expression instruction, followed by the arguments, one of which is an array of nodes in the expression tree.

The structure of the i.asph file is detailed below. Each field is 4 bytes, unless stated otherwise. All the offsets are from the beginning of the i.asph file, unless stated otherwise.

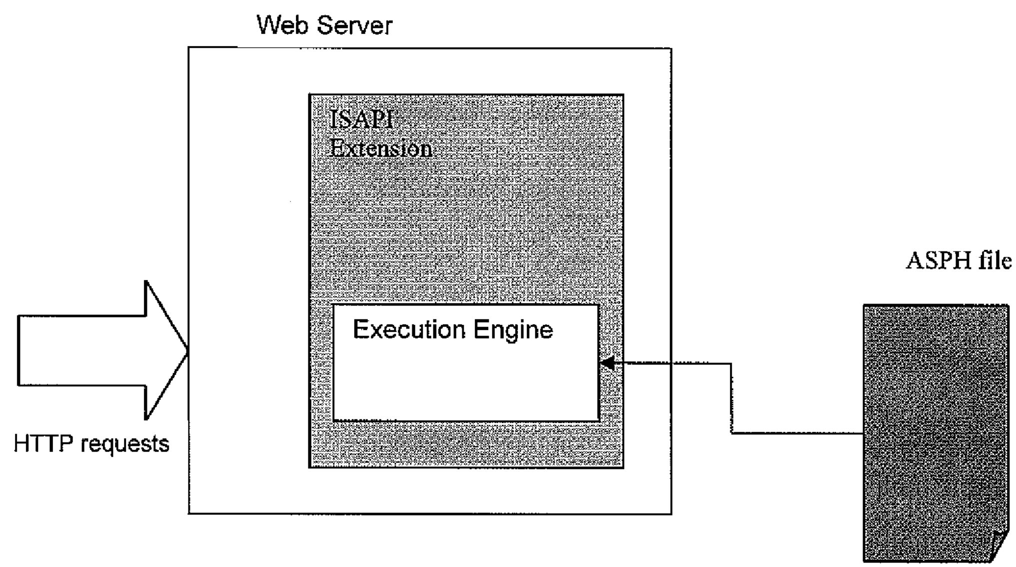
<table>
<tr><td colspan="2">Timestamp</td></tr>
<tr><td colspan="2">"asph"</td></tr>
<tr><td colspan="2">Number of Internal Variables</td></tr>
<tr><td colspan="2">Number of Internal Built-In Variables</td></tr>
<tr><td colspan="2">Number of Files</td></tr>
<tr><td>Number of Files</td><td>Offset of File Name Table</td></tr>
<tr><td>Number of WC Entries</td><td>Offset of WC Name Table</td></tr>
<tr><td>Number of SO Entries</td><td>Offset of SO Name Table</td></tr>
<tr><td>Number of ISAPI Variables</td><td>Offset of ISAPI Variables Name Table</td></tr>
<tr><td>Number of Internal Variables</td><td>Offset of Internal Variables Name Table</td></tr>
<tr><td colspan="2">Number of Languages</td></tr>
<tr><td></td><td></td></tr>
</table>

Language Code 1	Offset of File Table 1
......	..
Language Code n	Offset of File Table n
File Index 1	
Length (2 bytes)	Filename 1 (variable)
....	
.....	....
File Index n	
Length(2 bytes)	Filename n (variable)
WC Variable Index 1	
Length (2 bytes)	WC Variable 1 Name (variable)
....	
....	.....
WC Variable Index n	
Length(2 bytes)	WC Variable n Name(variable)
SO Variable Index 1	
Length (2 bytes)	SO Variable 1 Name (variable)
....	
....	.....
SO Variable Index n	
Length(2 bytes)	SO Variable n Name(variable)
ISAPI Variable Index 1	
Length(2 bytes)	ISAPI Variable 1 name (variable)
....	
....	....
ISAPI Variable Index n	
Length(2 bytes)	ISAPI Variable n name (variable)
Internal Variable Index 1	
Length(2 bytes)	Internal Variable 1 name (variable)
.....	
....	....
Internal Variable Index n	
Length(2 bytes)	Internal Variable n name (variable)
Language Code 1	
Offset of File 1	
....	
Offset of File n	

Language Code 1
Code for File 1 (variable)
.....
Code for File n (variable)
....
....
....
Language Code 1
Offset of File 1
....
Offset of File n
Language Code 1
Code for File 1 (variable)
.....
Code for File n (variable)

Note: WC and SO are some special types of variables which need a reverse lookup (name to internal variable index) that are used at Hotmail. In general, it can be assumed that a certain amount of reverse lookup will be needed by any system such as when an external party needs to input values for internal variables through HTTP.

Diagrams & Flowcharts:



Date of Conception: 7/1/2002 **Date Reduced to Practice: 10/152002**

Prior/Planned Disclosure (conferences & publications/betas/on sale/RTM/and please note if under NDA and the relevant dates):

Additional required info that is helpful and will save you time to provide now

Related Art:

Competitors/Competitive Products/Possible Licensees/Market Niche: High performance Web applications

Relationship to Standards:

MS-Technology Independent Implementations:

Attach related documents and links here.